

Balisage: The Markup Conference 2009 **Proceedings**



TEI Feature Structures as a Representation Format for Multiple Annotation and Generic XML Documents

Jens Stegmann

Bielefeld University

[<jens.stegmann@gmail.com>](mailto:jens.stegmann@gmail.com)

Andreas Witt

Institute for the German Language (IDS), Mannheim

[<witt@ids-mannheim.de>](mailto:witt@ids-mannheim.de)

Balisage: The Markup Conference 2009

August 11 - 14, 2009

Copyright © 2009 by the authors. Used with permission.

How to cite this paper

Stegmann, Jens, and Andreas Witt. "TEI Feature Structures as a Representation Format for Multiple Annotation and Generic XML Documents." Presented at Balisage: The Markup Conference 2009, Montréal, Canada, August 11 - 14, 2009. In *Proceedings of Balisage: The Markup Conference 2009*. Balisage Series on Markup Technologies, vol. 3 (2009). DOI: 10.4242/BalisageVol3.Stegmann01.

Abstract

Feature structures are mathematical entities (rooted labeled directed acyclic graphs) that can be represented as graph displays, attribute value matrices or as XML adhering to the constraints of a specialized TEI tag set. We demonstrate that this latter ISO-standardized format can be used as an integrative storage and exchange format for sets of multiple annotation XML documents. This specific domain of application is rooted in the approach of multiple annotations, which marks a possible solution for XML-compliant markup in scenarios with conflicting annotation hierarchies. A more extreme proposal consists in the possible use as a meta-representation format for generic XML documents. For both scenarios our strategy concerning pertinent feature structure representations is grounded on the XDM (XQuery 1.0 and XPath 2.0 Data Model). The ubiquitous hierarchical and sequential relationships within XML documents are represented by specific features that take ordered list values. The mapping to the TEI feature structure format has been implemented in the form of an XSLT 2.0 stylesheet. It can be characterized as exploiting aspects of both the push and pull processing paradigm as appropriate. An indexing mechanism is provided with regard to the multiple annotation documents scenario. Hence, implicit links concerning identical primary data are made explicit in the result format. In comparison to alternative representations, the TEI-based format does well in many respects, since it is both integrative and well-formed XML. However, the result documents tend to grow very large depending on the size of the input documents and their respective markup structure. This may also be considered as a downside regarding the proposed use for generic XML documents. On the positive side, it may be possible to achieve a hookup to methods and applications that have been developed for feature structure representations in the fields of (computational) linguistics and knowledge representation.

Table of Contents

Introduction

Feature Structures

Feature Structures in a Nutshell

The TEI Tag Set for Feature Structures

Representation and Transformation

Representation of XML Documents via TEI Feature Structures

Aspects of the XSLT Implementation of the Transformation

Summary and Outlook

Appendix A. Appendix: Result Document for the Annotation Data Examples

Introduction

As the title suggests, this contribution describes aspects of the use of a certain representation format ("TEI Feature Structures") with regard to a specific domain of application ("Multiple Annotation") and also concerning a second, much more general kind of scenario ("Generic XML Documents").

TEI P5 (Burnard and Bauman, 2007) compliant encodings of feature structures, which we refer to as *TEI feature structures* in this article, will receive much of our attention. Figure 1 shows a simple example: the encoding of a certain feature structure F_1 . F_1 serves to characterize a specific class of linguistic entities here, namely nominal phrases of the third person singular kind.

Figure 1: TEI Encoding of a Feature Structure F_1

```
<fs>
  <f name="CAT">
    <symbol value="np" />
  </f>
  <f name="AGR">
    <fs>
      <f name="NUM">
        <symbol value="sing" />
      </f>
      <f name="PER" />
        <symbol value="third" />
      </f>
    </fs>
  </f>
</fs>
```

There are two features on the top-level of F_1 : CAT with its value np and AGR with an associated complex value, which is a feature structure itself. This latter embedded structure consists of the feature-value pairs NUM with value sing and PER with value third. We will return to the theme of encoding F_1 below ("Feature Structures in a Nutshell"). Since we will use the same example there, it will be possible to compare different syntaxes for the display of feature structures in a straightforward way. We do not delve into details connected with the XML syntax exemplified in Figure 1 here, since this will be the topic of another part of this article ("The TEI Tag Set for Feature Structures"). In the rest of this introductory section, we shall try to shed some light upon the two application domains that have been mentioned above.

The more specific scenario consists in the integrative representation of annotation documents along the approach of *multiple annotations* (Witt, 2004). The multiple annotations approach marks a possible solution with regard to the markup of overlapping structures. Linguists, e.g., do often encounter XML-related problems, when they try to annotate a common core of linguistic data according to different levels of linguistic analysis (phonology, morphology, syntax, semantics, and pragmatics). The most straightforward way of marking things up might involve the incorporation of crossing edges. Such, however, is prohibited on grounds of XML's foundational tree structure. It can be argued that such configurations of data with conflicting hierarchies require a different kind of data structure, i.e., a *multi-rooted tree* ((Carletta et al., 2003), (Wörner et al., 2006) and (Witt et al., 2007)). A multi-rooted tree consists of several trees that span over the same data leaves. The multiple annotations approach now proposes to mark up each description level / tree as a document instance in its own right. This allows for each document to consist of well-formed XML, the modeling of alternative annotations is possible, the levels can be

viewed separately, and new levels can be added at any time (Witt, 2004). However, such documents may seem to be somewhat unrelated and independent of each other. Witt therefore proposes to regard the primary textual data, which have to be identical across all such annotation documents, as the defining implicit link between them. Of course, it would be desirable to bring such implicit linkages forward as explicit ones. This can be done, e.g., during the course of a transformation to an adequate representation format. We intend to show that the ISO-standardized TEI tag set for the representation of feature structures can be such a representation format. Pros, cons and alternative strategies with respect to *overlapping structures* are discussed in the pertinent literature, compare (DeRose, 2004), (Sperberg-McQueen, 2007) and (Carletta et al., 2007) for an overview.

Besides the different stages of the TEI recommendations ((Sperberg-McQueen and Burnard, 1994), (Sperberg-McQueen and Burnard, 2001) and (Burnard and Bauman, 2007)), at least one alternative proposal concerning the *encoding of feature structures as SGML/XML markup* has been brought forward in the literature (Sailer and Richter, 2001). However, to the best of our knowledge, no one has yet discussed the question how a representation in the opposite direction could look like, i.e., *encoding SGML/XML markup documents as feature structures*. We will come up with an original answer to this question, as we succeed concerning the more specific goal of finding a way to represent sets of multiple annotation documents as TEI feature structures. Feature Structures can be regarded as a general type of data structure and there may be specific advantages associated with their use as a meta-representation format. We will speculate about related aspects in the last section of this paper.

The structure for the rest of this article looks as follows. In the next section ("Feature Structures"), we characterize feature structures as mathematical entities and introduce three syntaxes for means of visualization and encoding: graph displays, attribute value matrices and the pertinent TEI tag set. Ways to represent XML documents as TEI feature structures and aspects of the XSLT-implemented transformation from multiple annotation and generic XML documents to the integrative TEI feature structure format are discussed in the next section ("Representation and Transformation"). Finally, we summarize our findings, take up some loose ends from the previous sections and discuss the relative advantages and disadvantages of representations in terms of TEI feature structures in the last section ("Summary and Outlook") of this contribution.

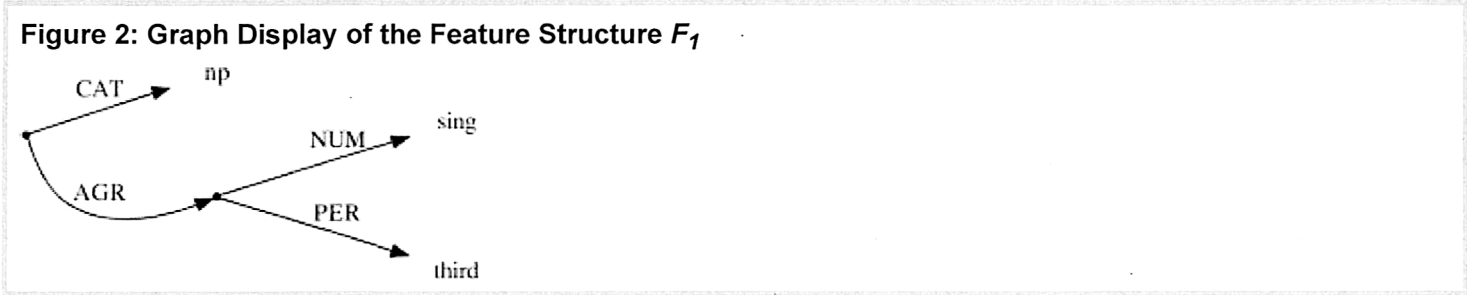
Feature Structures

Feature Structures in a Nutshell

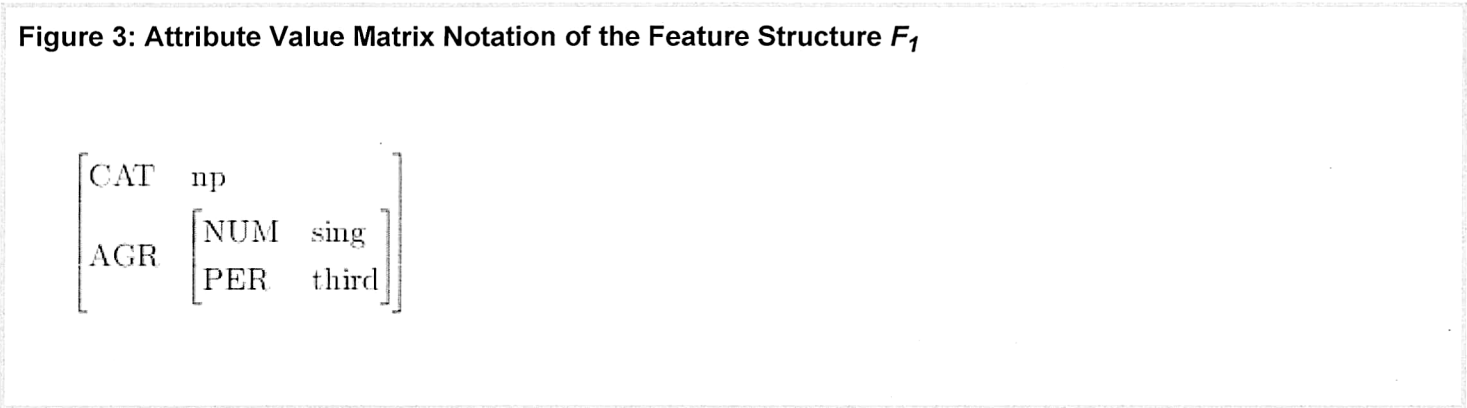
Feature structures are a common means of representation in formal linguistic theory.^[1] Their use is most prominent in certain variants of generative grammar (Shieber, 1986)^[2], but not constrained to the syntactic level of analysis, e.g., there are linguistic applications in phonology, morphology, semantics and pragmatics, too. Furthermore, feature structures can be characterized as a general purpose data structure (ISO24610, 2006) with possible applications in the vast field of knowledge representation. Hence, their usefulness is by no means constrained to linguistic investigations alone.

From a mathematical stance, there are at least two perspectives on feature structures (Shieber, 1986). On the one hand, a feature structure can be construed as a partial function from a set of *features* to a set of *values*. The value associated with a certain feature can be either *atomic*, e.g., a specific symbolic value as `element` or a binary value like `true`, or it may be *complex*. The latter means that it can be a full-blown feature structure itself or it may be of a *collection value* type like a set or a list of, again, possibly complex values. We will come upon numerous examples below. Due to the availability of complex values, feature structures can embed other feature structures in value position and, hence, provide a considerable degree of representational articulateness. Note that there will be no significance to the order of features that are located on the same hierarchical level within a feature structure. Another mathematical perspective derives from graph theory and leads to the characterization of feature structures as rooted labeled directed (acyclic)^[3] graphs. Graphs (Diestel, 2005) are mathematical entities that consist of sets

of nodes and edges. We can think of the edges of a graph as connecting its nodes. Graphs can be depicted in an intuitively appealing way as diagram displays. The labeled edges represent the features, the leaf nodes represent the atomic values, and the inner nodes, if any, represent the complex values of a feature structure. Figure 2 is an example *graph display* of the feature structure F_1 , compare Figure 1 in the preceding section for the TEI counterpart.



There is an alternative to the visualization of feature structures as graph displays. It consists in the use of attribute value matrices.^[4] In *attribute value matrix* notation, the features are written to the left of their associated values and there are brackets that indicate the scope of the (sub-)feature structure(s) involved.^[5] Figure 3 shows F_1 in attribute value matrix notation, compare Figure 1 and Figure 2 above for the TEI- and graph display counterparts. Concerning the forthcoming examples in this article, we will only use the TEI format and the attribute value matrix notation.



Feature structures list correct information and only correct information, but they do not necessarily contain all the correct information with regard to a specific object, i.e., they may be of a *partial* nature.^[6] Partiality can be a good thing, since it allows for feature structures to capture generalizations via the underspecification of certain properties.

When features have identical values, there are two scenarios to consider: the values can be either type- or token-identical. If the values are merely type-identical, we can characterize them as being independent of one another. A hypothetical change to one of the values would have no effect on the other values involved. However, in case of token-identity the features are associated with one and the same value token and, hence, are dependent on it. A change to the token would affect all the features that reference it. This latter scenario of token-identity is also called coreference, *structure sharing* or reentrancy. In attribute value matrix notation, it can be indicated by means of co-indexed boxes that either act as a referring place-holder in value position or they may be written before a certain value token and such all occurrences of the index within the feature structure are bound to that value. We will come upon an example in the next subsection of this article, cf. Figure 5 below. At the graph display level, we would use edges that lead into one and the same node in order to indicate structure sharing.

An important operation upon feature structures is unification (Shieber, 1986). The foundational idea is fairly simple and can be sketched as follows: the result of the unification of compatible feature structures is the most general feature structure that contains all the information of the unified feature structures. Technically, unification

is defined via the auxiliary concept of subsumption. *Subsumption* implements an intuitive concept of specificity and wealth of information among feature structures. We define that a feature structure F' subsumes a feature structure F'' if F' contains a subset of the information in F'' (Shieber, 1986). Alternatively, we may say that F' carries less information than F'' or that F' is more general than F'' . Subsumption is a partial order on the set of feature structures, since feature structures may be incompatible with each other. Now, we can define the *unification* of two feature structures F and G , if any, to be the most general feature structure H , such that F subsumes H and G subsumes H . If the feature structures to be unified are incompatible, we say that the unification fails. A related operation that works in the opposite direction is generalization. This operation is the dual of unification. We can define the *generalization* of two feature structures F and G to be the most specific feature structure E , such that E subsumes F and E subsumes G . Unlike unification, generalization cannot fail. In the worst case, the result will be the empty feature structure $[]$ that subsumes every feature structure.

It should be noted that feature structures can be *typed* (Carpenter, 1992). However, neither the present state of the representations nor the implemented transformation that we describe in this paper does make use of typed feature structures, so we won't go into details regarding that topic here.

The TEI Tag Set for Feature Structures

The TEI tag set for the representation of feature structures has been a part of the TEI Guidelines since version *P3* (Sperberg-McQueen and Burnard, 1994). Building on the *P4* version (Sperberg-McQueen and Burnard, 2001), an ISO standard (ISO24610, 2006) was adopted by ISO TC37 SC4 and also implemented in the current *P5* version (Burnard and Bauman, 2007) that we use here.

The foundational XML elements that are needed in order to encode feature structures are `fs` for feature structures and `f` for features. The content of an `fs` element consists of a sequence of feature-value specifications. A feature-value specification is encoded using an element of type `f` for the feature and the element content of `f` for the associated value. The details look as follows. Every `f` element has an attribute `name` for its feature name. The representation of the associated value of a feature depends on the exact type of the value involved. Atomic values of type `binary`, `symbol` or `numeric` are realized via a `value` attribute on a respective child element of `f` that corresponds to the actual value type. For example, `f` may have a child element `binary` which has a `value` attribute that provides the desired parameter. If the value is of the `string` type, however, the value is encoded in a slightly different form, i.e., as the literal element content of a respective `string` child element of `f`.

Complex values of the feature structure kind are encoded by means of `fs` elements, of course. However, there is also another class of complex values: these are the collection values of the `list`, `set` and `bag` type. Such collections of values are indicated via `vColl` elements that have an `org` attribute whose value specifies the respective collection type, i.e., whether it is a `bag`, a `set` or a `list`. The content of a `vColl` element consists of a succession of values of any kind.

Figure 4: TEI Feature Structure F_2 : Structure Sharing and Collection Values

```
<fs>
  <f name="F">
    <vColl org="list">
      <vLabel name="a">
        <fs>
          <f name="I">
            <symbol value="a"/>
          </f>
          <f name="J">
            <symbol value="b"/>
          </f>
        </fs>
      </vLabel>
    </vColl>
  </f>
  <vLabel name="b">
    <fs>
      <f name="K">
        <symbol value="c"/>
      </f>
      <f name="L">
        <symbol value="d"/>
      </f>
    </fs>
  </vLabel>
</fs>
```

```

    </f>
  </fs>
</vLabel>
</vColl>
</f>
<f name="G">
  <vLabel name="a"/>
</f>
<f name="H">
  <vColl org="set">
    <vLabel name="b"/>
    <fs>
      <f name="M">
        <symbol value="e"/>
      </f>
      <f name="N">
        <symbol value="f"/>
      </f>
    </fs>
  </vColl>
</f>
</fs>

```

There is a special element in order to indicate cases of structure sharing: the `vLabel` element. It either contains a value token as its element content or it occurs as a placeholder which indicates reference to an elsewhere specified value token. Each `vLabel` element has an associated `name` attribute. The value of the `name` attribute corresponds to the index of a tagged box in attribute value matrix notation, see below. This mechanism allows for various structure sharing configurations within a single feature structure. Figure 4 (TEI-based representation) and Figure 5 (attribute value matrix notation) display the same abstract example feature structure F_2 in different notation formats and exemplify the themes of structure sharing and collection values.

Figure 5: Attribute Value Matrix for F_2 : Structure Sharing and Collection Values

$$\left[\begin{array}{l} F \left\langle \boxed{a} \left[\begin{array}{cc} I & a \\ J & b \end{array} \right], \boxed{b} \left[\begin{array}{cc} K & c \\ L & d \end{array} \right] \right\rangle \\ G \quad \boxed{a} \\ H \quad \left\{ \boxed{b}, \left[\begin{array}{cc} M & e \\ N & f \end{array} \right] \right\} \end{array} \right]$$

There are three top-level features in F_2 : F , G , and H . All of them are associated with complex values. F has a list collection value, which is encoded using angle brackets at the attribute value matrix level, G has a feature structure as its value, and H has a set collection value that is indicated using curly brackets in Figure 4. The first list value of F and the complex value of G are co-indicated. The same holds for the second list value of F and the firstly notated set member of H .

Representation and Transformation

Representation of XML Documents via TEI Feature Structures

Both feature structures and XML documents can be regarded from the perspective of graph theory (Diestel, 2005). XML documents are specimen of ordered trees, while feature structures are merely unordered directed acyclic graphs. This holds because of the possibility of structure sharing within feature structures and because there is no order imposed among features of the same level within feature structures. So, the task of representing XML documents as feature structures seems to involve a transformation from a more rigidly structured representation format to a less rigidly structured one. Specifically, we have to find a way to represent the ordered *sequential*


```

    </vLabel>
  </f>
  <f name="REST">
    <fs>
      <f name="FIRST">
        <vLabel name="5">
          <symbol value="n"/>
        </vLabel>
      </f>
      <f name="REST">
        <symbol value="*null*" />
      </f>
    </fs>
  </f>
</fs>
</f>
</fs>
</f>
</fs>
</f>
</fs>
</f>
</fs>
</f>
</fs>
<f name="TIER1">
  ...
</f>
<f name="TIER2">
  ...
</f>
</fs>

```

This way of representation is displayed in Figure 8 in an abridged TEI feature structure format that shows the top-level feature geometry of the structure :^[8] DATA contains a representation of only the textual characters of the document adhering to the FIRST/REST scheme discussed above. Furthermore, each character is associated with its own index in order to allow for structure sharing references to it from other parts of the feature structure. We provide an index for every character in order to allow for arbitrarily specific levels of annotation with respect to the common textual data. The numbered TIER features contain the specific information of the annotation levels. Each one represents the information of one of the multiple annotation documents involved. The implicit link between the different levels is made explicit by means of structure sharing. Therefore, there will be plenty of references to the common data characters from within the different TIER features of the document. However, this is not an explicit part of the example display in Figure 8 due to space considerations.^[9] The binding of indexes to certain values is shown within the DATA feature, but the reference to such values is hidden within the abridged TIER levels of the document. However, those parts and the connections provided by the structure sharing mechanism can be inspected in Figure 9 that shows the attribute value matrix notation. Unlike its TEI counterpart, this display is complete and probably a bit easier to follow. It also displays the mechanics of the representation of the hierarchical relationships. They find expression via CONTENT features, whose values contain the representation of the subordinated document parts, e.g., the content of an element. The mechanisms for the representation of the hierarchical and the sequential relationships have to be combined as appropriate. This means that CONTENT will have a list value and the respective position within that list will reflect the sequential order among the dominated document parts.

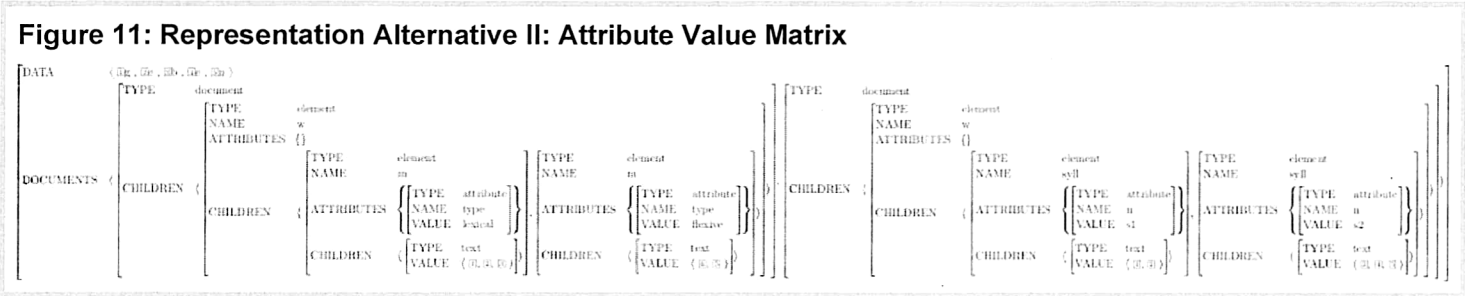
Figure 9: Representation Alternative I: Attribute Value Matrix


```

</vColl>
</f>
</fs>

```

On the top level, this representation consists of a `DATA` feature and a `DOCUMENTS` feature. Both features take complex values of a collection type, i.e., lists of values. Concerning `DATA`, we now have a flat list representation with little internal structure. This format can be built in an easier way as compared to the more structured variant. The move to this format is possible, since the TEI Guidelines provide this kind of notational sugar for values of the collection kind.^[10] The `DOCUMENTS` feature also makes use of this kind of list notation and embeds the representation of the different annotation documents as a flat list of respective feature structures. Note also that there is just one such top-level feature now, compare the different numbered `TIER` features in representation alternative I. If there is only one annotation document to process, the list will contain only one corresponding feature structure, of course. Figure 11 is a complete display of the attribute value matrix for our example data.



This way of representation takes a stance that is based on the *XQuery 1.0 and XPath 2.0 Data Model (XDM)* and, hence, the representations will be predestined for processing in an XSLT 2.0 context. We use the attributes which are provided by the XDM in order to represent the different node kinds within an XML document, starting from the very root. The node kinds that are distinguished are: document, element, attribute, namespace, commentary, processing instruction and text nodes. Every occurrence of a node is represented as a feature structure with features as appropriate for the node kind involved. The type of a node is indicated via the `TYPE` feature for nodes of all kinds. Hierarchical relations are represented via the `CHILDREN` feature for document- and element nodes. Order among the children nodes is encoded by the position within a sequence, since `CHILDREN` takes a collection value of the list kind. Element nodes and attribute nodes have `NAME` features, attribute and text nodes have `VALUE` features. Furthermore, each element node has an `ATTRIBUTES` feature that takes a set value, since attributes are unordered. The semantics associated with the different feature-value pairs should be straightforward. All in all, this approach allows for a very systematic representation regime across the different parts of an arbitrary XML document instance. Unlike the older approach, every feature structure which is embedded below the `DOCUMENTS` top-level feature now represents a certain node at the XML tree model level. However, it also has to be noted that our feature structure representations of XML documents tend to grow very fast with the size of the input document, which, however, seems to be true for all approaches based on TEI feature structures due to the modeling as feature structure and also the retranslation to XML involved.^[11]

If the input to the transformation program does not consist of multiple annotation documents, but rather of one or several arbitrary XML documents, which do not share identical primary data, an integrative representation of such documents will still be build in a similar way. However, there will be no indexing mechanism incorporated and so no implicit links will be made explicit.

Aspects of the XSLT Implementation of the Transformation

The program `xmls2avm.xsl` that implements the transformation to the TEI feature structure format was written with multiple annotation documents in mind. Nevertheless, it is robust enough to provide a result document if the input documents to the transformation fail the test of primary data identity or if there is only one document to be

transformed. Such kind of robustness marks a necessary condition for the program to be useful within the generic XML realm.

The program was written in *XSLT 2.0* and uses certain features of the new XSLT version. For example, data typing is used for at least some of the parameters and variables involved and, most importantly, we exploit the extended functionalities and constructs that are grounded on the XDM tree model. XSLT 2.0 comes with support for multiple output documents, but the multiple input documents that are needed here still have to be provided via a sort of workaround: a call of the `document()`-function to a post-processed representation of a stylesheet parameter. The latter contains a list of secondary input documents that has to be assigned by the user when invoking the transformation program from the command line. Several further stylesheet parameters are provided in order to parameterize certain aspects of the transformation process and to determine peculiarities of the desired representation format. Most of this is optional, however, since there are defaults for the relevant parameters. An example stylesheet parameter is `$firstrestRepr`: it influences the way how lists are represented. If it is set to `true`, then lists will be represented in the recursively structured way that has been introduced as our historically older representation alternative I in the previous section. If it is set to `false`, however, then lists will be represented according to the newer flat representation alternative II that exploits the notational sugar provided by the TEI guidelines. The parameter is set to `false` as a default.

Although we decided that we would not include detailed comments on the whole stylesheet^[12], we do provide three illustrative template examples below. These will be the templates for document nodes (in default mode), attributes and text. Besides these, the full stylesheet also contains templates for document nodes (in secondary mode), elements, processing instructions, comments. Furthermore, there are named templates for the processing of nested sequences and for the processing of nested sequences with regard to namespaces, as well as many additional parameters and variables defined.

We begin our discussion with the template for *document nodes in default mode*, i.e., the mode that is used at the start of the transformation without further ado. The template shown in Figure 12 will be applied to the document node of the primary input document at the start of the transformation.

Figure 12: Template for Document Nodes in Default Mode

```
<xsl:template match="document-node()" mode="#default">
  <xsl:variable name="children" select="node()"/>
  <xsl:variable name="textnodes" select="//text()"/>
  <fs>
    <xsl:if test="$dataIdentity and $dataRepr">
      <f name="DATA">
        <vColl org="list">
          <xsl:for-each select="str:characters($primaryString)">
            <vLabel name="{position()}">
              <string>
                <xsl:value-of select="."/>
              </string>
            </vLabel>
          </xsl:for-each>
        </vColl>
      </f>
    </xsl:if>
    <f name="DOCUMENTS">
      <vColl org="list">
        <fs>
          <f name="TYPE">
            <symbol value="document"/>
          </f>
          <f name="CHILDREN">
            <xsl:choose>
              <xsl:when test="$firstrestRepr">
                <xsl:choose>
                  <xsl:when test="$children">
                    <xsl:call-template name="SequenceProcessing">
                      <xsl:with-param name="seq" select="$children"/>
                    </xsl:call-template>
                  </xsl:when>
                  <xsl:otherwise>
                    <symbol value="*null*" />
                  </xsl:otherwise>
                </xsl:choose>
              </xsl:when>
              <xsl:otherwise>
                <vColl org="list">
                  <xsl:apply-templates select="$children"/>
                </vColl>
              </xsl:otherwise>
            </xsl:choose>
          </f>
        </fs>
      </vColl>
    </f>
  </vColl>
</vColl>
```

```

        </xsl:otherwise>
      </xsl:choose>
    </f>
  </fs>
  <xsl:apply-templates select="$docRoots" mode="secondary"/>
</vColl>
</f>
</fs>
</xsl:template>

```

The template starts with the definition of variables that can be referenced within the scope of the template. Most of the other templates in the stylesheet use such template variables, too. Then the first `fs` element of the target representation is inserted. This will be the outer frame for all the result markup that is created during the transformation. The two usual top-level features for a feature structure representation of XML documents are `DATA` and `DOCUMENTS`, compare our discussion of representation alternative II in the previous section. It is possible to drop even the `DATA` feature and go with the `DOCUMENTS` feature on the top-level of the feature structure alone. This possibility has been parameterized using `$dataRepr`, i.e., the user may decide whether he wants a `DATA` feature at the top-level or not. Furthermore, a variable named `$dataidentity` has been defined on the global stylesheet level. This variable implements a test for the identity of the primary textual data of all the input documents involved. Now, if `DATA` shall be present and the test result concerning textual data identity is positive, then the feature will be inserted into the result and be given a list value. The content of that list will be construed as follows: we iterate over all textual characters of our primary input document. For each character, we insert index markup (`vLabel`) with a numerical index attribute according to the position value of the respective character. Furthermore, the index will be bound to the character value whereas the latter is framed by a `string` element to indicate its value type. Next is the obligatory `DOCUMENTS` feature. It will take a list of feature structures, i.e., a list of `fs` elements. Those will represent the input documents, respectively.

In what follows in this template, we build the representation for the primary input document. The corresponding job for the other input documents, if any, will have to be done by the template for document node kinds in secondary mode. The two features appropriate for document nodes are `TYPE` and `CHILDREN`. Concerning `TYPE`, its value will be `<symbol value="document"/>` obviously. The value of `CHILDREN`, however, is more complicated and has to be determined via a series of conditional constructs. Firstly, it depends on whether the list representation has been set to the older recursively structured kind (`$firstrestRepr`) or not. If list representations follow that approach, then it depends again on whether the document node has descendants or not. If he has none, we insert a value for the empty list (`*null*`). However, if there are descendant nodes to the document node, the further calculation of the list representation is taken over by a called template of the recursive kind named `SequenceProcessing`. This template is called with the sequence of the current document node's descendant nodes as a parameter. That template will build a recursively structured kind of list representation as appropriate. However, if the value of the parameter `$firstrestRepr` is set such that we will have the flat kind of list representation, which is the default, then markup for a collection of the list kind will be inserted. However, the content of that list will be determined by the result of applying templates to all the descendant nodes of the current document node. Thus, the content of the `fs` element for the current primary input document is complete and can be closed with the respective end tags. What remains to be computed is the markup for the other secondary input documents. Therefore, templates are applied to the members of `$docRoots`, which holds the document nodes of the secondary input documents in a sequence format. Note, that a mode (`secondary`) is used in the respective `apply-templates` instruction, so the present template will not fit and we avoid a repeated insertion of the initial framing markup for the outermost level of the feature structure representation, which is only included in the processing of the primary input document here.

The complete stylesheet can be characterized as exploiting aspects of both *the push and the pull processing paradigm* (Tennison, 2005), like most stylesheets of a considerable size and complexity do, whereas the focus is

shifting in different parts of the stylesheet. In a similar vein, it can be classified as implementing different *stylesheet design patterns* (Kay, 2008). For example, the buildup of the initial target feature structure tends to be of the pull type or rather navigational, to use Kay's concept. This, however, shifts towards a more push- or rule-oriented approach, which helps to fill up the missing parts of the initial structure by applying templates to the descendants of the current node. Appropriate templates are provided for each specific node kind against the background of the XDM. Certain aspects, e.g., the buildup of the older FIRST/REST list structures have been realized in a computational way recursively via calls to named templates with parameters as their arguments. We shall look at a recipient template of the push- or rule-oriented style of processing next in Figure 13. It is the template for the processing of *attribute nodes*, whose application will be initiated from within the template for the processing of element nodes.

Figure 13: Template for Attribute Nodes

```
<xsl:template match="attribute()" mode="#all">
  <fs>
    <f name="TYPE">
      <symbol value="attribute"/>
    </f>
    <f name="NAME">
      <string>
        <xsl:value-of select="node-name(.)"/>
      </string>
    </f>
    <f name="VALUE">
      <string>
        <xsl:value-of select="."/>
      </string>
    </f>
  </fs>
</xsl:template>
```

In comparison to the previous template for document nodes, this one is very straightforward. There are three features appropriate for feature structures that represent attribute nodes: these are TYPE, NAME and VALUE. The respective values are very easily determined. Readers who managed to follow through on our description of the previous template should have no problems with this one.

At the heart of the transformation of multiply annotated documents is the indexing of the single characters and the reference mechanism that exploits these indexes. It is dependent on the relative position of characters with respect to the other characters of the document. Those values can be used as numerical indexes since they are bound to be constant across all the documents that pass a test of primary data identity. However, it has to be stressed that the computational cost of implementing this functionality can be considerable for large input documents. Figure 14 shows the code which does the job: it is the template for *text nodes*.

Figure 14: Template for Text Nodes

```
<xsl:template match="text()" mode="#all">
  <xsl:variable name="currentRoot" select=""/>
  <fs>
    <f name="TYPE">
      <symbol value="text"/>
    </f>
    <f name="VALUE">
      <xsl:choose>
        <xsl:when test="$dataIdentity">
          <xsl:variable name="numberOfCharactersSoFar" as="xs:integer"
            select="sum(for $textnode in preceding::text() return string-length($textnode))"/>
          <vColl org="list">
            <xsl:for-each select="str:characters(string(.))">
              <vLabel name="{position() + $numberOfCharactersSoFar}">
                <xsl:if test="not($dataRepr) and $primary is $currentRoot">
                  <string>
                    <xsl:value-of select="."/>
                  </string>
                </xsl:if>
              </vLabel>
            </xsl:for-each>
          </vColl>
        </xsl:when>
        <xsl:otherwise>
          <string>
            <xsl:value-of select="."/>
          </string>
        </xsl:otherwise>
      </xsl:choose>
    </f>
  </fs>
```

```
</xsl:choose>
</f>
</fs>
</xsl:template>
```

There are two appropriate features for text nodes: `TYPE` and `VALUE`. The `TYPE` feature is set to the symbolic value `text`, of course. The procedure for determining the value of the feature `VALUE`, however, is much more complicated. This holds at least for multiple annotation documents, where the identity of the primary data is given (`$dataIdentity`). If this is not the case, we can just insert the value of the textual node as a whole. With regard to the data-identity scenario, however, we will proceed on a character by character basis with the help of an appropriately defined external function (`str:characters`) and calculate the appropriate index for each character. The interesting part of the calculation is done in the binding of the variable `$numberOfCharactersSoFar`. That result will be modulated by the relative position of each character with respect to the string value of the text node processed. If the user chose to go without the `DATA` feature on the top feature geometry level (`not($dataRepr)`) and if we are processing the primary input document (`$primary is $currentRoot`), not only the calculated indexes will be included in the list-valued result, but also the character values. Now that there is no specialized `DATA` feature, the indexes will be bound to their respective value tokens here.

Summary and Outlook

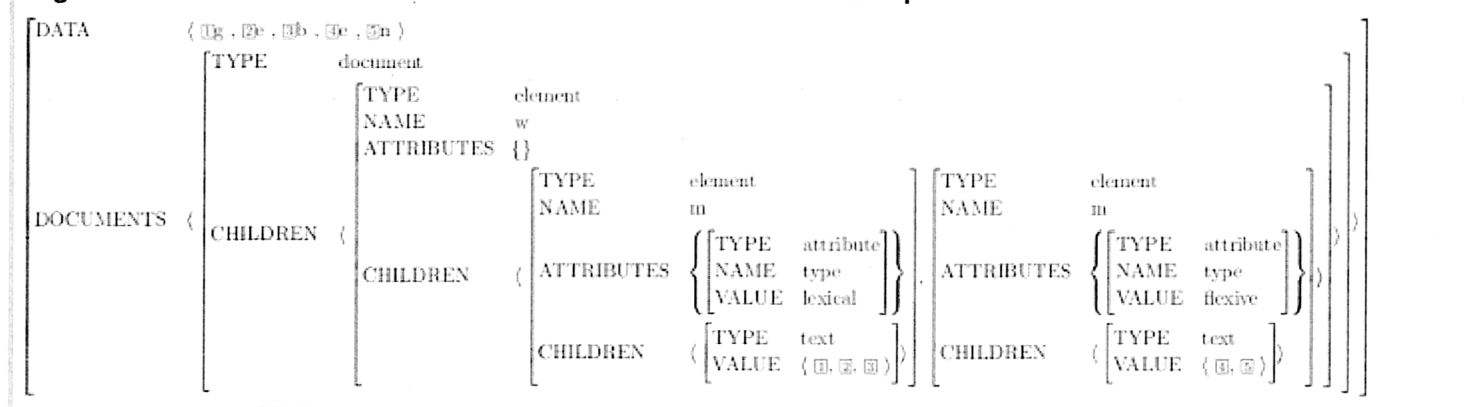
In the context of this article, we started by providing an informal introduction to feature structures and their encoding as proposed in the TEI P5 Guidelines. We continued to discuss aspects of the representation of multiple annotation documents as XML-encoded feature structures. Most of our pertinent remarks are also correct concerning the representation of generic XML documents. It is rather just the indexing mechanism that is lost for that more general domain. Furthermore, we characterized the implemented XSLT stylesheet that was written in order to bring about the transformation from multiply annotated or generic XML documents to TEI-based feature structure representations. In the remainder of this article, we will take up some loose ends and speculate about possible advantages and disadvantages that may be connected with the format.

In comparison to alternative proposals like *XCONCUR* ((Hilbert et al., 2005), (Schonefeld and Witt, 2006)) and the *NITE XML* format (Carletta et al., 2003), the following advantages and disadvantages can be stated. Like *NITE XML*, but unlike *XCONCUR* documents, the TEI-based feature structure format is an XML format, which should count as a definitive plus in most contexts. Furthermore, like *XCONCUR*, but unlike the *NITE XML* representations, the proposed TEI feature structures are integrative in a strict sense of the word. What we mean is that all the distributed annotation information is made available within the context of a single document instance in which the implicit links have been made explicit. So, with regard to these two aspects, TEI feature structures seem to do quite well in comparison with the mentioned alternative formats, which lack in the one or the other way. However, there is also a big downside connected to them. The TEI feature structure representations grow very fast with the size of the input documents and their relative markup complexity, much faster than both rival formats.^[13] So serious doubts remain, whether this format can prevail in practical day-to-day-work if it is used for collections of large resource documents.

But are there any striking advantages that may be connected with the representation of XML documents in a feature structure format? Feature structures are a common data structure in linguistic theory and they play an important role in many implementations in computational linguistics. If the preferred representation format of computational linguists can be used, it may be possible to find a way to apply the processing tools that have been developed in that field and bridge the gap between the information given by annotations and the information contained in textual content. One may also speculate whether general operations on feature structures like *unification* and *generalization*, compare the section "Feature Structures in a Nutshell" above, may be applicable to

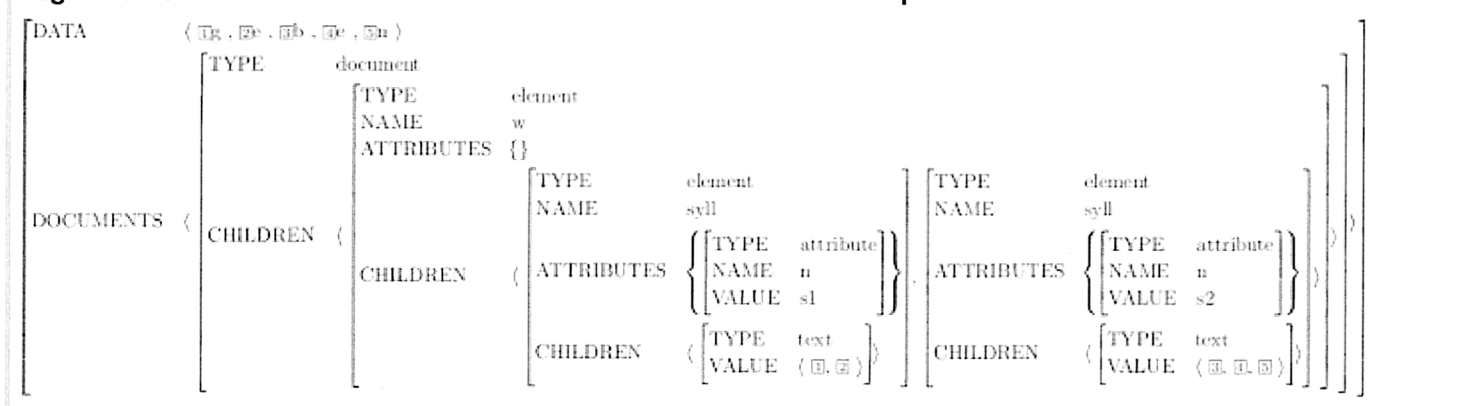
appropriately represented XML documents or linguistic corpora.

Figure 15: Attribute Value Matrix Notation of the Annotation Example 1



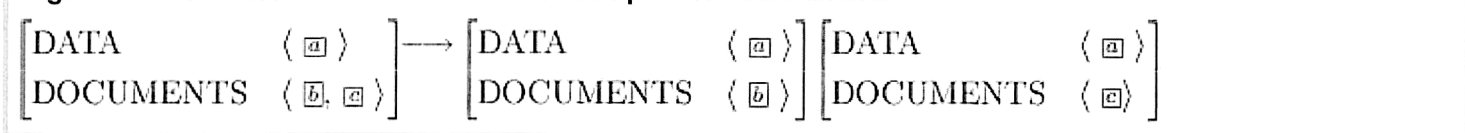
Compare Figure 15 and Figure 16. These are possible TEI feature structures for the simple linguistic annotation examples that we have used before. Unlike Figure 11, which is an integrative representation of both example documents, each figure here displays the representation of just one annotation document. These examples will help us to explore some of the issues involved.

Figure 16: Attribute Value Matrix Notation of the Annotation Example 2



As before, we have to consider two broad scenarios: operations among multiply annotated documents and operations among generic XML documents. The main difference between both has to do with the values of the `DATA` feature.^[14] For multiple annotation, the values of `DATA` will be identical and the respective features can, hence, be unified. However, for generic XML documents the `DATA` values will almost always be different. Hence, they usually won't unify. And even multiply annotated documents will run into problems when it comes to the value of the `DOCUMENTS` feature slot, compare Figure 15 and Figure 16. So the bare unification of complete representations does not seem to work out for either class of documents.

Figure 17: Rule that uses Unification for Multiple Annotation Data



However, there is a way how unification may be put to use with regard to respective representations, but in a somewhat different way. It works analogously to the way in which unification is put to use in *linguistic rules* in unification-based grammars. We do not unify the whole representations, but only parts of it in accordance to a rule, which directs how to build a bigger structure from smaller structures (or vice versa, this is a question of procedural interpretation). Structures that are coindexed within a rule have to be unified when the rule is applied. In line with this, e.g., our annotation examples (on the right hand side of the rule) can be projected to a bigger structure (on the left hand side of the rule) as displayed in Figure 17. For generic XML documents, a rule like Figure 18 might work.

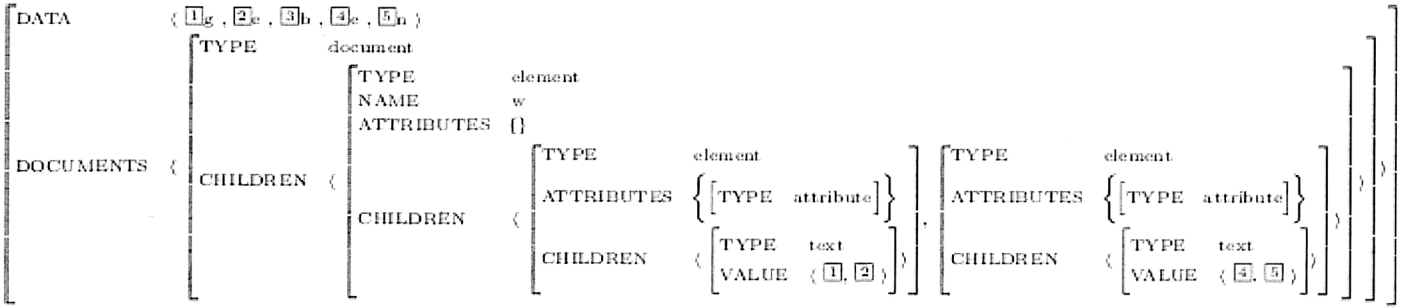
Figure 18: Rule that uses Unification for Generic XML Documents

$$\left[\begin{array}{l} \text{DATA} \\ \text{DOCUMENTS} \end{array} \begin{array}{l} \langle \underline{a}, \underline{b} \rangle \\ \langle \underline{a}, \underline{d} \rangle \end{array} \right] \rightarrow \left[\begin{array}{l} \text{DATA} \\ \text{DOCUMENTS} \end{array} \begin{array}{l} \langle \underline{a} \rangle \\ \langle \underline{a} \rangle \end{array} \right] \left[\begin{array}{l} \text{DATA} \\ \text{DOCUMENTS} \end{array} \begin{array}{l} \langle \underline{b} \rangle \\ \langle \underline{d} \rangle \end{array} \right]$$

For another perspective on the unification of XML documents compare (Witt et al., 2005).

There is also a second general operation on feature structures: generalization. Unlike unification, generalization cannot fail. And indeed, generalization can be put to use concerning our examples here. The result indicates what is common to both representations and is shown in Figure 19.

Figure 19: Generalization of the Annotation Data Examples 1 and 2



One of the anonymous reviewers of this paper stated that (s)he thinks that its strength is "as a sort of thought experiment that has not provided quite the breakthrough that was hoped for it; yet interesting things have been learned and observed." This is not too far off from our own perspective. Although we were able to show that this and that can be done, at least in principle---as things stand, we do not think that it is likely that TEI feature structures will turn out to be the silver bullet for the representation of linguistic annotations or generic XML documents. Our respective representations grow too fast and isn't yet clear, whether good and sensible use can be made of the general operations on feature structures open to us now, i.e., whether the potential advantages can override the disadvantages connected to it. But it seems that there are at least some open questions that remain to be investigated. For example, perhaps we could come up with a different way of representing XML documents in terms of TEI feature structures as compared to our current representation practice and see if that helps in any way. Going with typed feature structures might be a worthwhile thing to try. However, we think that the prospects are not too good, since the foundational issue of complex modeling and retranslating to XML would basically stay the same and it seems that this is quite an overhead to cope with. Therefore, finally, we will at least mention a different direction that has been encouraged by the very same reviewer mentioned above. (S)he advised to step away from the TEI-ness of the present approach in order to investigate the prospects of bare feature structures, e.g., in the sense of an implemented library, with respect to the issues at hand.

Appendix A. Appendix: Result Document for the Annotation Data Examples

```
<?xml version="1.0" encoding="UTF-8"?>
<fs>
  <f name="DATA">
    <vColl org="list">
      <vLabel name="1">
        <string>g</string>
      </vLabel>
      <vLabel name="2">
        <string>e</string>
      </vLabel>
      <vLabel name="3">
        <string>b</string>
      </vLabel>
      <vLabel name="4">
        <string>e</string>
      </vLabel>
      <vLabel name="5">
        <string>n</string>
      </vLabel>
    </vColl>
  </f>
  <f name="DOCUMENTS">
```



```

<vColl org="list">
  <fs>
    <f name="TYPE">
      <symbol value="document"/>
    </f>
    <f name="CHILDREN">
      <vColl org="list">
        <fs>
          <f name="TYPE">
            <symbol value="element"/>
          </f>
          <f name="NAME">
            <string>w</string>
          </f>
          <f name="ATTRIBUTES">
            <vColl org="set"/>
          </f>
          <f name="CHILDREN">
            <vColl org="list">
              <fs>
                <f name="TYPE">
                  <symbol value="element"/>
                </f>
                <f name="NAME">
                  <string>m</string>
                </f>
                <f name="ATTRIBUTES">
                  <vColl org="set">
                    <fs>
                      <f name="TYPE">
                        <symbol value="attribute"/>
                      </f>
                      <f name="NAME">
                        <string>type</string>
                      </f>
                      <f name="VALUE">
                        <string>lexical</string>
                      </f>
                    </fs>
                  </vColl>
                </f>
              </fs>
            </vColl>
          </f>
          <f name="CHILDREN">
            <vColl org="list">
              <fs>
                <f name="TYPE">
                  <symbol value="text"/>
                </f>
                <f name="VALUE">
                  <vColl org="list">
                    <vLabel name="1"/>
                    <vLabel name="2"/>
                    <vLabel name="3"/>
                  </vColl>
                </f>
              </fs>
            </vColl>
          </f>
          <f name="CHILDREN">
            <vColl org="list">
              <fs>
                <f name="TYPE">
                  <symbol value="element"/>
                </f>
                <f name="NAME">
                  <string>m</string>
                </f>
                <f name="ATTRIBUTES">
                  <vColl org="set">
                    <fs>
                      <f name="TYPE">
                        <symbol value="attribute"/>
                      </f>
                      <f name="NAME">
                        <string>type</string>
                      </f>
                      <f name="VALUE">
                        <string>flexive</string>
                      </f>
                    </fs>
                  </vColl>
                </f>
              </fs>
            </vColl>
          </f>
          <f name="CHILDREN">
            <vColl org="list">
              <fs>
                <f name="TYPE">
                  <symbol value="text"/>
                </f>
                <f name="VALUE">
                  <vColl org="list">
                    <vLabel name="4"/>
                    <vLabel name="5"/>
                  </vColl>
                </f>
              </fs>
            </vColl>
          </f>
        </fs>
      </vColl>
    </f>
  </fs>
</vColl>

```

```

<f name="TYPE">
  <symbol value="document"/>
</f>
<f name="CHILDREN">
  <vColl org="list">
    <fs>
      <f name="TYPE">
        <symbol value="element"/>
      </f>
      <f name="NAME">
        <string>w</string>
      </f>
      <f name="ATTRIBUTES">
        <vColl org="set"/>
      </f>
      <f name="CHILDREN">
        <vColl org="list">
          <fs>
            <f name="TYPE">
              <symbol value="element"/>
            </f>
            <f name="NAME">
              <string>syll</string>
            </f>
            <f name="ATTRIBUTES">
              <vColl org="set">
                <fs>
                  <f name="TYPE">
                    <symbol value="attribute"/>
                  </f>
                  <f name="NAME">
                    <string>n</string>
                  </f>
                  <f name="VALUE">
                    <string>s1</string>
                  </f>
                </fs>
              </vColl>
            </f>
            <f name="CHILDREN">
              <vColl org="list">
                <fs>
                  <f name="TYPE">
                    <symbol value="text"/>
                  </f>
                  <f name="VALUE">
                    <vColl org="list">
                      <vLabel name="1"/>
                      <vLabel name="2"/>
                    </vColl>
                  </f>
                </fs>
              </vColl>
            </f>
          </fs>
        </vColl>
      </f>
      <f name="CHILDREN">
        <vColl org="list">
          <fs>
            <f name="TYPE">
              <symbol value="element"/>
            </f>
            <f name="NAME">
              <string>syll</string>
            </f>
            <f name="ATTRIBUTES">
              <vColl org="set">
                <fs>
                  <f name="TYPE">
                    <symbol value="attribute"/>
                  </f>
                  <f name="NAME">
                    <string>n</string>
                  </f>
                  <f name="VALUE">
                    <string>s2</string>
                  </f>
                </fs>
              </vColl>
            </f>
            <f name="CHILDREN">
              <vColl org="list">
                <fs>
                  <f name="TYPE">
                    <symbol value="text"/>
                  </f>
                  <f name="VALUE">
                    <vColl org="list">
                      <vLabel name="3"/>
                      <vLabel name="4"/>
                      <vLabel name="5"/>
                    </vColl>
                  </f>
                </fs>
              </vColl>
            </f>
          </fs>
        </vColl>
      </f>
    </fs>
  </vColl>
</f>

```

References

- [(Burnard and Bauman, 2007)] Burnard, L. and Bauman, S. *TEI P5: Guidelines for Electronic Text Encoding and Interchange*. Text Encoding Initiative, 2007
- [(Carletta et al., 2003)] Carletta, J.; Kilgour, J.; O'Donnell, T.; Evert, S. and Voormann, H. *The NITE Object Model Library for Handling Structured Linguistic Annotation on Multimodal Data Sets*. In: Proceedings of the EACL Workshop on Language Technology and the Semantic Web (3rd Workshop on NLP and XML, NLPXML-2003), 2003
- [(Carletta et al., 2007)] Carletta, J.; DeRose, S.; Durusau, P.; Piez, W.; Sperberg-McQueen, C. M.; Tennison, J. and Witt, A. *International Workshop on Markup of Overlapping Structures*. In: Usdin, B. T. (ed.) Proceedings of Extreme Markup Languages 2007, 2007
- [(Carpenter, 1992)] Carpenter, B. *The Logic of Typed Feature Structures: With Applications to Unification Grammars, Logic Programs and Constraint Resolution*. Cambridge University Press, 1992
- [(DeRose, 2004)] DeRose, S. *Markup Overlap: A Review and a Horse*. In: Usdin, B. T. (ed.) Proceedings of Extreme Markup Languages 2004, 2004
- [(Diestel, 2005)] Diestel, R. *Graph Theory*. Springer, 2005
- [(Hilbert et al., 2005)] Hilbert, M.; Schonefeld, O. and Witt, A. *Making CONCUR work*. In: Usdin, B. T. (ed.) Proceedings of Extreme Markup Languages 2005, 2005
- [(ISO24610, 2006)] 24610-1:2006, I. *Language Resource Management -- Feature Structures -- Part 1: Feature Structure Representation*. International Organization for Standardization, 2006
- [(Kay, 2008)] Kay, M. *XSLT 2.0 and XPath 2.0 Programmer's Reference*. Wrox Press Ltd., 2008
- [(NLM, 2008)] *Custom Metadata Group*. In: Journal Archiving and Interchange Tag Set Tag Library version 3.0, Version of November 2008.
- [(Pollard and Sag, 1994)] Pollard, C. and Sag, I. *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, 1994
- [(Sailer and Richter, 2001)] Sailer, M. and Richter, F. *Eine XML-Kodierung für AVM-Beschreibungen*. In: Lobin, H. (ed.). Sprach- und Texttechnologie in digitalen Medien: Proceedings der GLDV-Frühjahrstagung 2001. BOD - Books on Demand, 2001, 161-168
- [(Schonefeld and Witt, 2006)] Schonefeld, O. and Witt, A. *Towards validation of concurrent markup*. In: Usdin, B. T. (ed.). Proceedings of Extreme Markup Languages 2006, 2006
- [(Shieber, 1986)] Shieber, S. M. *An Introduction to Unification-based Approaches to Grammar*. CSLI Publications, 1986
- [(Sperberg-McQueen and Burnard, 1994)] Sperberg-McQueen, C. M. and Burnard, L. *TEI Guidelines for Electronic Text Encoding and Interchange (TEI P3)*. Text Encoding Initiative, 1994
- [(Sperberg-McQueen and Burnard, 2001)] Sperberg-McQueen, C. M. and Burnard, L. *Guidelines for Electronic Text Encoding and Interchange (TEI P4)*. Text Encoding Initiative, 2001
- [(Sperberg-McQueen, 2007)] Sperberg-McQueen, C. M. *Representation of overlapping structures*. In: Usdin, B. T. (ed.) Extreme Markup Languages 2007, 2007
- [(Tennison, 2005)] Tennison, J. *Beginning XSLT 2.0: From Novice to Professional*. Apress, 2005
- [(Witt, 2004)] Witt, A. *Multiple Hierarchies: New Aspects of an Old Solution*. In: Usdin, B. T. (ed.) Proceedings of Extreme Markup Languages 2004, 2004
- [(Witt et al., 2005)] Witt, A.; Goecke, D.; Sasaki, F. and Lungen, H. *Unification of XML Documents with Concurrent Markup*. Literary and Linguistic Computing, 2005, 20, 103-116, doi:10.1093/llc/fqh046
- [(Witt et al., 2007)] Witt, A.; Schonefeld, O.; Rehm, G.; Khoo, J. and Evang, K. *On the Lossless Transformation of Single-File Multi-Layer Annotations into Multi-Rooted Trees*. In: Usdin, B. T. (ed.). Proceedings of Extreme Markup Languages 2007, 2007

- [(Witt et al., 2009)] Witt, A.; Rehm, G.; Hinrichs, E.; Lehmberg, T. and Stegmann, J. *SusTEInability of Linguistic Resources through Feature Structures*. Literary and Linguistic Computing, 2009, 24, 363-372, doi:10.1093/llc/fqp024
- [(Wörner et al., 2006)] Wörner, K.; Witt, A.; Rehm, G. and Dipper, S. *Modelling Linguistic Data Structures*. In: Usdin, B. T. (ed.). Proceedings of Extreme Markup Languages 2006, 2006

-
- [1] There are equivalent structures in other environments, too, as one of our anonymous reviewers remarked. Compare the National Library of Medicine's *custom-meta structures* (NLM,2008), for example.
- [2] Namely unification-based grammars, whose name derives from the most important operation on feature structures, i.e., unification.
- [3] Some formalizations of feature structures allow cycles and it can also be argued that cyclic structures may be needed for the representation of certain phenomena as the liar's paradox ("This statement is false.").
- [4] Some linguistic theories use different notations for (total) models vs. (partial) descriptions. For example, HPSG (Pollard and Sag, 1994) uses graph displays for models and AVMs for descriptions.
- [5] Feature names are usually capitalized on grounds of a notational convention.
- [6] HPSG theoreticians (Pollard and Sag, 1994) draw a distinction between *feature structures*, which can be characterized as total objects in the sense of containing all the relevant specifications with respect to the objects they are a model of, and *feature structure descriptions*, which are partial descriptions of feature structures. From this perspective, feature structures and feature structure descriptions belong to different theoretical realms (model vs. formalism). We will not delve deeper into this discussion here and continue with our usage of the term feature structure for partial objects also.
- [7] Unless there is no rest sequence and we have reached the end of the sequence already.
- [8] It would be nice to have something like a specialized document grammar regarding the finer details of the representations that we propose in this article. One of our anonymous reviewers encouraged us to give *Feature System Declarations* (Burnard and Bauman, 2007) for the TEI feature structures. However, it seems that TEI FSDs are reserved for typed feature structures and the present state of our work here makes use of untyped feature structures. Since we may well choose to make the switch to typed representations in the future (in a way, the new representation scheme below has been designed to make the switch easier), it will be a good idea to take up on that proposal in a respective update. For the moment, we can at least validate our documents against TEI feature structure schemas generated via the TEI ROMA tool (<http://www.tei-c.org/Roma/>).
- [9] Note that a complete representation of the above annotation data examples in TEI format, but according to the newer representation standard that will be discussed below in this subsection, can be found in Appendix A.
- [10] In terms of features and values alone, respective structures still have to be realized by FIRST/REST-like structured representations as introduced above. The format provided by the TEI is a shorthand for that.
- [11] As one of our anonymous reviewers remarked, it would be interesting to investigate the prospects and the performance of bare feature structures for our purposes and see whether and how much better they can perform as compared to the TEI-serialized feature structures that are the focus of the present paper.
- [12] This decision was made on grounds of space considerations, since this is a rather long paper already. Some anonymous reviewers would have liked to see the whole stylesheet included. Others shared our perspective that examples suffice here.
- [13] XCONCUR seems to be leanest in this respect.
- [14] For the sake of the argument, we will presume that there will be a DATA feature on the top-level of all TEI feature structures. The stylesheet does not force this, though.

Jens Stegmann

<jens.stegmann@googlemail.com>

Bielefeld University

Jens Stegmann studied linguistics, psychology and computer science at Bielefeld University. Parts of this paper deal with aspects of his Master thesis.

Andreas Witt

<witt@ids-mannheim.de>

Institute for the German Language (IDS), Mannheim

Witt received his Ph.D. in Computational Linguistics and Text Technology from the Bielefeld University in 2002 (dissertation title: Multiple Informationsstrukturierung mit Auszeichnungssprachen. XML-basierte Methoden und deren Nutzen für die Sprachtechnologie).

After graduating in 1996, he started as a researcher and instructor in Computational Linguistics and Text Technology. He was heavily involved in the establishment of the minor subject Text Technology in Bielefeld University's Magister and B.A. program in 1999 and 2002 respectively. After his Ph.D. in 2002 he became an assistant lecturer, still at the Text Technology group in Bielefeld. In 2006 he moved to Tübingen University, where he was involved in a project on "Sustainability of Linguistic Resources" and in projects on the interoperability of language data. Since 2009 he is senior researcher at "Institut für Deutsche Sprache" (Institute for the German Language) in Mannheim.

Witt is and was a member of several research organizations, amongst them the TEI Special Interest Group on overlapping markup, for which he was involved in the writing of the latest version of the chapter "Multiple Hierarchies", which is included in TEI-Guidelines P5.

Witt's main research interests deal with questions on the use and limitations of markup languages for the linguistic description of language data.

Balisage Series on Markup Technologies